### 参考

<http://www.yindaheng98.top/%E5%9B%BE%E5%BD%A2%E5%AD%A6/%E6%8A%95%E5%BD%B1%E5%92%8C%E5%85%89%E6%A0%85%E5%8C%96.html#%E9%BD%90%E6%AC%A1%E5%9D%90%E6%A0%87%E5%92%8C%E9%BD%90%E6%AC%A1%E5%90%91%E9%87%8F>

书籍：Mathematics for 3D Game Programming and Computer Graphics

### Lecture04

Camera的定义：position，lookat direction，up direction

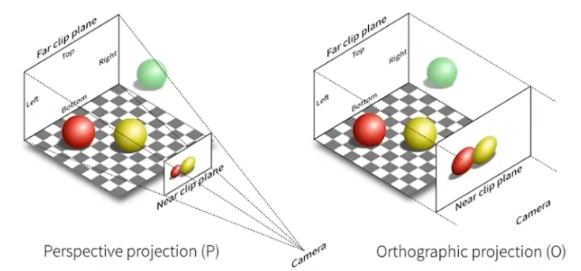
约定熟成：相机放在000原点，up为y轴，看着-z轴

View/camera transformation：把相机移动到上面约定的位姿，也可以理解为世界坐标系转到相机坐标系下，但是相机坐标系有约定，y轴向上，看向-z轴，在此基础上再做Projection

Perspective projection / orthographic projection

透视投影：透视投影转换到cuboid（M\_persp->ortho），再做正交投影

正交投影：透视投影后视锥体范围就对应一长方体，视锥体各平面就是长方体的各平面，正交投影就是在透视投影后把这个长方体内的点全部缩放到x,y,z∈[−1,1]3*x*,*y*,*z*∈[−1,1]3的立方体区域内以方便光栅化计算。



View Transformation：View Transformation 矩阵将物体顶点在世界空间下的坐标转换为视图空间下的坐标（相机坐标系）。(注意和计算机视觉的世界坐标系转相机坐标系不同，这里的转换是相机看向-z，up为y轴正向)

Projection Transformation：Projection Transformation 矩阵将物体顶点在视图空间的坐标转换为裁剪空间的坐标。

什么是正交投影矩阵M\_ortho。 也就是标准化正交投影，压缩到-1，1的立方体

什么是透视投影转正交投影M\_persp->ortho。

实际流程是先透视投影转正交投影，然后标准化正交投影

### Projection Matrix推导

<https://yconquesty.github.io/blog/ml/nerf/nerf_ndc.html#background>

Mathematics for 3D Game Programming and Computer Graphics

坐标系x轴向右，y轴向上，按看向-z轴推导，假设n>0, f>0，近平面为z=-n, 远平面为z=-f

M矩阵乘以(Px,Py,Pz,1）会变换到这里的z是坐标为负的，这里n和f都是正的

求出Mortho，-f<=Pz<=-n，但是NDC需要的是把near平面映射到-1, far平面映射到+1，所以需要对z轴进行反转（等于换到NDC左手系）

注意：Mathematics for 3D Game Programming and Computer Graphics书中给了一步到位推导，就是直接映射到-1,1去

### 光栅化Rasterization

光栅就是screen的意思，光栅化=画在屏幕上

假设已经得到了[-1,1]的立方体

Viewport变换。转换xy plane:

A math equation with numbers and lines

AI-generated content may be incorrect.

所谓光栅化，就是确定每个点是否处于物体内，如果处于显示物体的颜色，否则不显示。这个过程专业点叫sampling

MVP变换：

model transform：把object的local坐标系转换到世界坐标系下

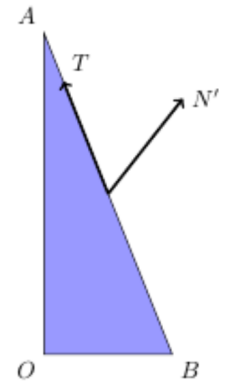
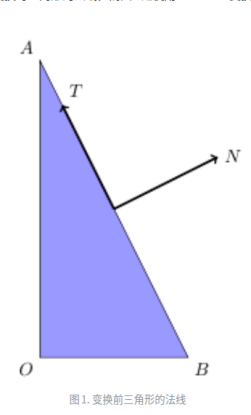
view transform：把object从世界坐标系转换到相机坐标系下

projection transform：投影到NDC空间下（clip space）

viewport transform：变换到屏幕下

法线怎么变换？

vertex可以利用MVP来变换，但是法线不能利用直接乘以n来计算



将MV看作一个整体，下面推导法线变换到view space下

假设为切线向量，切线向量可以由某两个点P2 P1得到

（注意T为方向向量，齐次写法最后一个值为0）

所以可以利用MV来变换切线向量

假设有一个3x3矩阵G，把法线变换后依然满足，这里实际只用了MV的左上角3x3子矩阵

把点乘改为矩阵乘法形式，方便推导

又因为

所以

### 抗锯齿antialiasing

先做模糊再做sampling，也就是低通滤波。

MSAA，一个像素内部不止采样一次，而是多个采样点。判断覆盖率：看这个像素的采样点是否在三角形（几何体）内。最终一个像素的颜色 = 所有采样点的结果加权平均。

**混叠现象**：高于采样带宽的成分会“折叠”到低频区，表现出来就是**边缘呈现阶梯状的锯齿**。

### Z-buffer algorithm（光栅化的z怎么处理）

遍历所有triangles，更新triangles涉及到的像素的深度，保留深度最小值

### Shading

Blinn-Phong Reflectance Model。经验模型。光照简单分为三类：specular highlights，diffuse reflection， ambient lighting，也就是镜面高光，漫反射，环境光。

环境光：Blinn-Phong并不精确描述环境光，只用一个非常简单的近似来表示即为环境光产生的亮度=材质对环境光的反射系数\*环境光强度（常量constant）

Lambert漫反射：漫反射指的是光从一定角度入射之后，向各个方向均匀反射的效果。A diagram of a light source

AI-generated content may be incorrect.

注：

为入射光强度，表示离光源越远，入射光强度会衰减，表示法线向量点乘入射方向（注意是表面指向光源），也就是只和有关，同时注意，漫反射强度和相机观测位置无关。

A diagram of a reflection of light

AI-generated content may be incorrect.

镜面反射：求出法线向量和half vector的夹角的cosine值来判断相机观测能接收到多少镜面反射的光强度，这里类比初中物理。用half vector而不直接计算入射光反射向量的原因是half vector计算更简单，指数P让结果更narrow

A diagram of a blue x and a black background

AI-generated content may be incorrect.

上图是完整计算公式

**Shading frequencies**

Flat shading: 对每个面进行Blinn-Phong模型计算，将该结果应用于整个面

Gouraud Shading：对每个三角形vertex进行shading计算，有了顶点的shading结果之后，对三角形内部的每一个点进行重心坐标插值。

那么怎么求vertex法向量？把该vertex周围的面的法线向量求均值归一化

A diagram of a hexagon with arrows and lines

AI-generated content may be incorrect.

Phong Shading：对每个pixel进行shading，如何求每个pixel的法线向量？利用重心

A diagram of a line with lines and dots

AI-generated content may be incorrect.

A gold teapot with a handle

AI-generated content may be incorrect.

A diagram of a process

AI-generated content may be incorrect.

### 重心坐标

可以得到方程

写成矩阵形式

克拉默法则：对于线性方程组，如果A是一个可逆矩阵(det A!=0)，则方程组有解，其中，其中表示列向量c取代了A的第i列后得到的矩阵

根据克拉默法则，我们可以写出

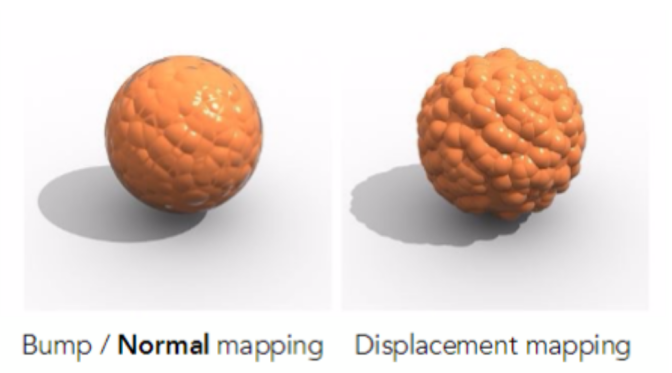
### 纹理映射

环境光贴图。将所有方向的光记录下来保存，有球面保存，但是投影到平面的时候，极点会出现扭曲。可以使用立方体来保存，但是会导致额外的计算，因为立方体无法直接获得某一方向的光照。

纹理影响shading。

Bump Mapping：并不真的改动三角形网格，而是进行法线的扰动从而改变shading结果。缺点：由于并不是真的改变实际网格，会导致shadow和轮廓依然是原始形状。

Displacement mapping：会使得三角形网格真的移动



### Geometry

隐式：level sets / distance functions

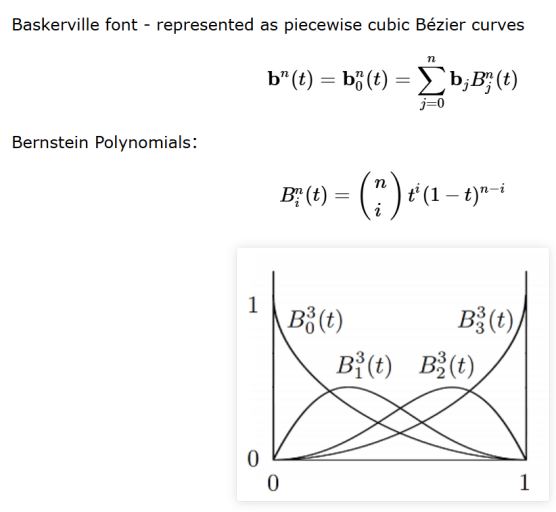
显式：point cloud，polygon mesh，subdivision

隐式，比如球体的方程x2+y2+z2=1

显式，点云/三角面/**参数映射（bezier 曲线）**

Bezier 曲线。

分段Bezier 曲线。怎么保证分段Bezier曲线连接处光滑，简单可以使用：前一个曲线的后两个控制点和后一个曲线的前两个控制点共线，并且和中间的点等距



Mesh subdivision，网格细分。

Mesh simplification网格简化

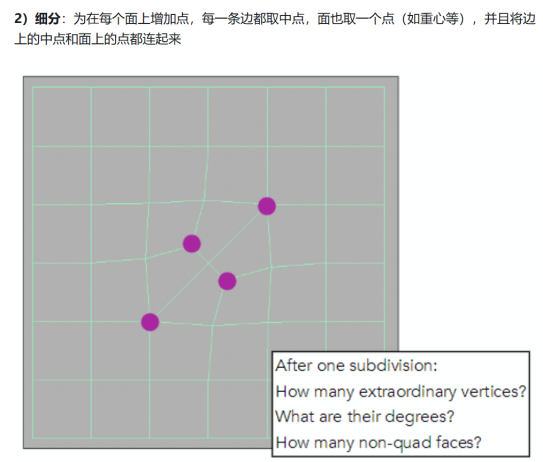
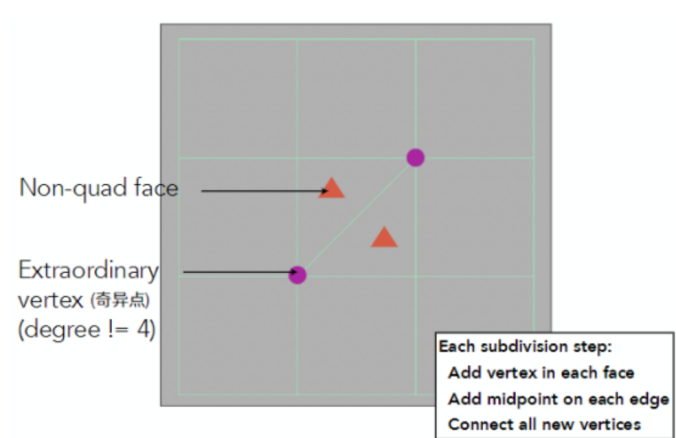
Mesh regularization网格正规化，让三角形更像正三角形

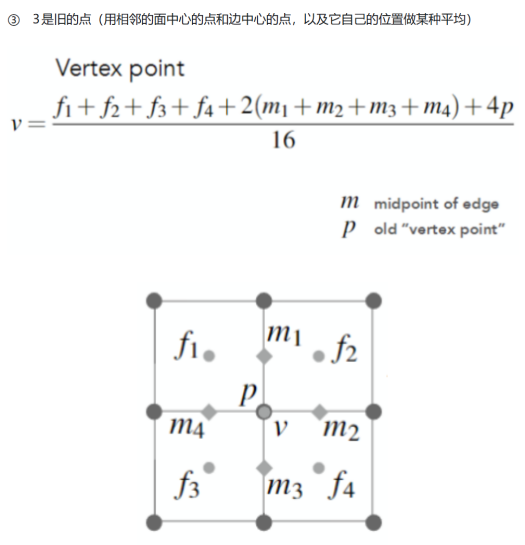
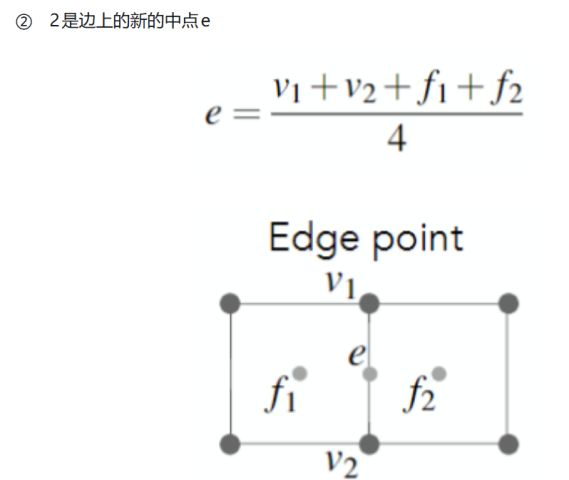
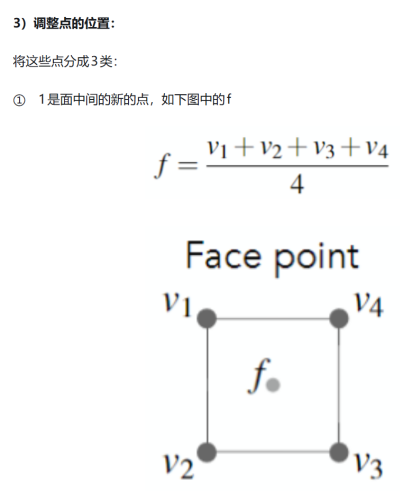
Loop subdivision。1. 连接三条边中点继而拆成4个三角形。2. 根据规则调整新/旧三角形vertex。但是Loop subdivision只能应用于三角形。



Catmull-Clark Subdivision (General Mesh)。可以应用于一般情况，四边形和三角形同时存在。

Quad face / non-quad face，extraordinary vertex奇异点：degree不为4的点





### Shadow Mapping

通俗讲就是，点光源会产生一个zbuffer图，相机光栅化也有一个深度图，对于相机光栅看到的每个点，投影回点光源，比较点光源的zbuffer图，如果深度一致，则说明相机和点光源都能看见，则不是阴影，如果zbuffer图更小，则说明点光源无法看到这个点，但相机可以看到，这个点为阴影。

缺点，shadow map本身有分辨率，应该为多大？1. Shadow map 太小会走样 2. 只能做硬阴影 3. 数值精度问题

### Ray Tracing

光栅化的局限性。无法很好的处理global effects，例如soft shadows，glossy reflection，indirect illumination。

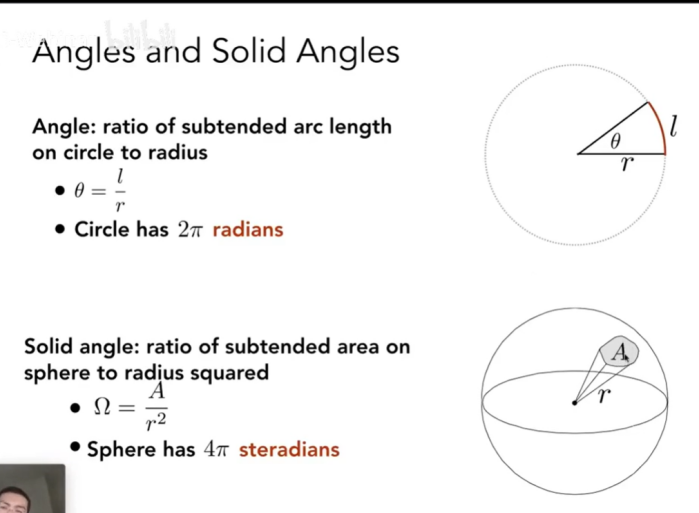
**Recursive (Whitted-Style) Ray Tracing如何求光线与表面的交点**

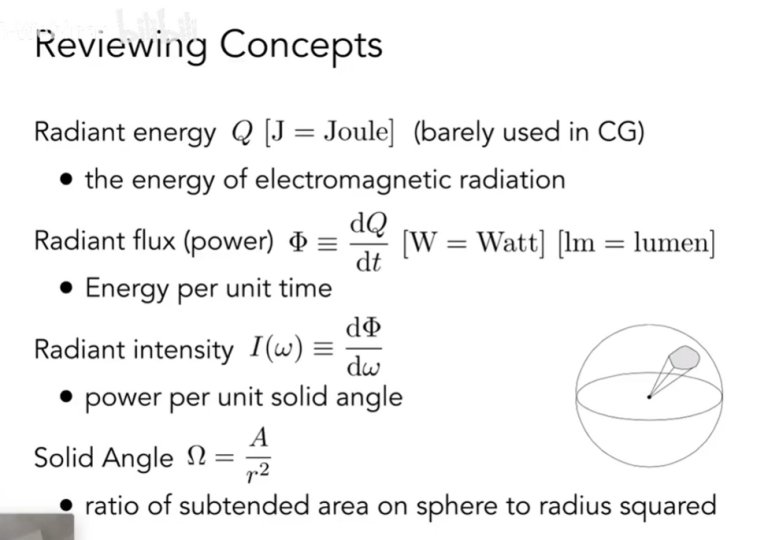
**对于隐式表面，直接求解方程判断。**

**对于光线与一个三角形求交点，分为两步，1. 求光线与平面的交点。2. 判断交点是否在三角形内。当然还有快速的做法：Möller Trumbore Algorithm，也就是利用重心坐标来判断。**

**对于光线与一个物体求交。简单的做法是遍历所有三角形来判断是否有交点，当然这非常慢。**

**利用Bounding Volumes来做加速，利用Axis-aligned Bounding Box (AABB)轴对齐包围盒，来加速判断，如果连box都没碰到则无需检查物体。**





Radiant辐射的

Radiant energy/power辐射能量，单位J

Radiant flux 辐射通量 单位W， 每个单位时间的能量

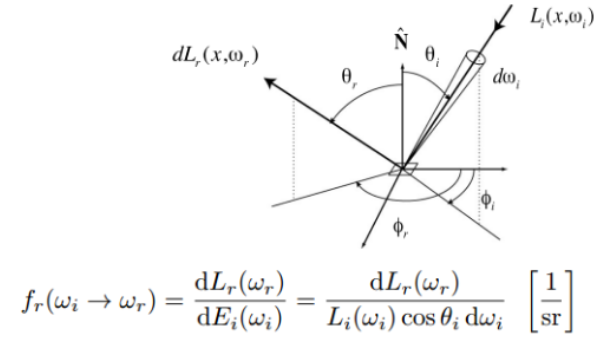
Radiant intensity辐射强度，每个单位时间每个单位立体角的能量

立体角指的是球上的面积和半径raidus平方的比值

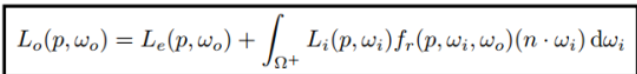
Irradiance: total power received by area dA（dA收到的所有的能量）

Radiance: power received by area dA from “direction” dω（dA从某一个方向收到的能量）

**Bidirectional Reflectance Distribution Function(BRDF)双向反射分布函数。**一根光线打到一个漫反射物体，则会向四面八方射出，BRDF用来描述光线从某个方向进来，反射到某个方向上的能量应该是多少。

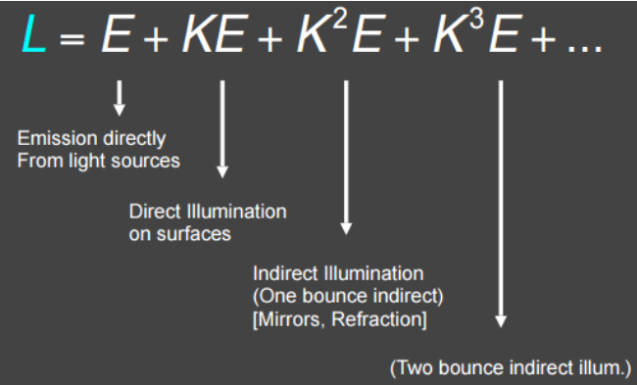


**The Reflection Equation（反射方程）**



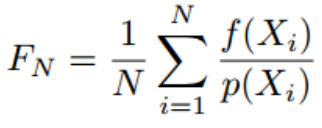
某个方向接收到的light等于，该物体在p点的自身发光，加上，对p点的半球积分累计所有入射光的强度乘以BRDR乘以

最终会写成（没有推导过程）



弹射一次叫做直接光照，弹射2次及以上为间接光照。直接光照+间接光照=全局光照。

蒙特卡洛积分。如果要对求积分，同时)随机变量x服从这个PDF

则积分可以近似成

**Whitted-Style Ray Tracing只追踪反射/折射，当光线打到漫反射物体时，光线就停了。**

1. **Whitted-Style Ray Tracing对镜面材质（specular）是对的，而对于glossy材质主要为镜面反射又有一部分漫反射。**
2. **Whitted-Style Ray Tracing在漫反射物体上停止传播，而实际漫反射也会传播光线（四面八方反射）**

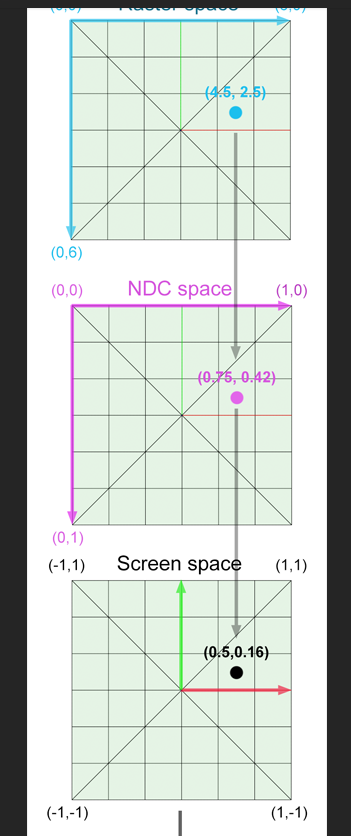
### 作业5

世界空间-相机空间-屏幕空间-NDC空间-栅格空间

之前的MVP变换是正向过程，现在生成ray是从右到左的过程，因为要根据像素位置来找到之前实际的camera space下的坐标，又由于相机原点为000,所以可以获得ray

<https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/generating-camera-rays.html>

在raster space取一个像素格子中心，转换到NDC space（注意这里的NDC space和之前raster章节定义的NDC不一样）



再进一步转换到screen space，此时认为image plane为z=-1

此时先考虑imageapectratio维持图像原始比例，然后乘以实际的fov产生的scale，就获得了z=-1上的实际xy坐标，其实这里也可以使用其他z=-2/-3,因为最终是为了获得一条射线，但是z=-1最简单。